Consistency and Recovery Control for Replicated Files

Danco Davcev*
Walter A. Burkhard

Computer Systems Research Group
Department of Electrical Engineering and Computer Sciences
University of California, San Diego
La Jolla, California 92093

Abstract

We present a consistency and recovery control scheme for replicated files. The purpose of a replicated file is to improve the availability of a logical file in the presence of site failures and network partitions. The accessible physical copies of a replicated file will be mutually consistent and behave like a single copy as far as the user can tell. Our recovery scheme requires no manual intervention. The scheme tolerates any number of site failures and network partitions as well as repairs.

1. Introduction

We present a consistency and recovery control scheme for replicated files. The purpose of a replicated file is to improve the availability of a logical file in the presence of site failures and network partitions. The accessible physical copies of a replicated file will be mutually consistent and behave like a single copy as far as the user can tell. Our goals are to provide a scheme for maintaining replicated files in a distributed environment that 1) will ensure mutual consistency at the level of individual file operations and 2) rapid updating of all accessible physical copies. We are interested in maintaining mutual consistency through network partitions and site failures.

The literature contains very few solutions to such problems. We will review the pertinent previous literature. We are unable to find any previous work that completely solves the recovery problem in the presence of network partitions and subsequent repairs.

In the true-copy token scheme [Mino82], tokens are used to designate the physical components that hold the current ‘‘logical component’’ values. Only copies designated by tokens can be accessed if network partitioning occurs. However, if these copies reside in a rarely used portion of the network, some transactions may be blocked. When the loss of all true-copies of some logical component is detected, a costly true-copy regeneration voting process must be undertaken.

The primary site approach [Alsb76] allows access only to the primary site partition block. If the primary site crashes, the system must elect an alternate primary site and this election requires a majority of the replicated copies.

Parker et al. developed an automatic conflict detection method for mutual inconsistency in distributed file systems. During network partitioning, multiple users can modify different copies of a replicated file. However, resolving the detected consistency conflicts is left to the user [Park83].

---

* Dr. Davcev is a visiting Fulbright Scientist.

A method for automatic conflict detection and transaction back-out is proposed in [Davi84]. Conflicts are detected from cycles in the so-called ‘‘precedence graph’’ and are resolved by transaction backout. At repair time, the transactions are rerun by the system or manually.

Garcia-Molina et al. [Garc83] present principles of an integration program for independently updating copies of a database during network partitioning. The Data-Patch tool, to aid database administrators in the integration of database copies, is discussed.

The consistency among replicated objects in JASMIN [Wilk84] is controlled in such way that one copy is designated as the ‘‘anchor’’ copy. The write operations during network partition are performed on all copies in the partition containing the anchor copy. However, the authors state that ‘‘failure of the anchor copy is a problem’’ and leave this recovery issue unresolved for network partitioning.

Voting is used in many algorithms in distributed systems to provide mutual exclusion [Thom79], [Giff79], [Skee82], [Garc82], [Segu79], [Mens80], [Elli83] and others. During network partitioning mutual consistency between replicated files is guaranteed and this is the principal advantage of voting over other consistency control schemes. Unfortunately, as has been mentioned recently by many authors [Park83], [Davi84], [Wilk84], [Barb84], if none of the partition groups have a majority of the total number of physical copies of a replicated file in the system, no data operation can occur anywhere. The vulnerability of voting schemes to this type of failure has been studied by Barbara and Garcia-Molina [Barb84] . In [Bern84] an algorithm for consistency control and recovery in replicated distributed databases is given which can handle detectable site failures.

We present a complete solution to the problem of consistency and recovery in the presence of network partitions and repairs. Moreover, our solution provides certain advantages beyond those of schemes previously proposed.

i) User or operator intervention is never required during recovery from network partitions.

ii) The mutual consistency of accessible physical copies within replicated files is guaranteed.

iii) The operations of our scheme are simple to state and program.

iv) The availability of the copies of a replicated file is considerably greater than in all previously published algorithms that guarantee mutual consistency.

v) The necessary quorum of copies for an update or read operation dynamically changes.

While the proposed scheme does have several advantages, there are some disadvantages to note as well.

i) Access to a replicated file can be momentarily restricted. In the worst such case (a very unlikely event), the user will be advised to close the file and continue at a later time.

ii) Data operations for a replicated file can be applied to at most one group of mutually connected sites.

The new recovery scheme provides file availability somewhat between that of other schemes. The availability is better than that of Thomas's majority consensus algorithm but worse than that of Parker's automatic conflict detection method. Many of the previous network partition recovery schemes have traded file availability for file consistency. These schemes allow virtually unlimited access during a network partition; however, manual intervention may be required during the recovery and reconciliation of the physical copies of a replicated file. We are quite willing to temporarily restrict access to a replicated file in order to maintain mutual consistency among the physical copies of a replicated file.

We wish to emphasize at this point the differences between our scheme and previously published schemes.

1) Our scheme is different from all fixed quorum consensus schemes because it does not need a majority of the total number of copies of a replicated file to perform update operations on the file. In our scheme the required quorum changes dynamically and depends only upon the number of current copies at the last failure.

2) In our scheme, operational sites have an equal role in satisfying the required quorum.

3) Our scheme can perform update operations with as few as two current accessible copies for replicated file consisting of an arbitrary number of copies.

4) Our scheme prevents conflicts among the physical copies of a replicated file rather than detecting conflicts after they have occurred.

5) Our scheme accommodates failures at any time, particularly during the execution of an operation on a replicated file.

2. Environment Model

We are utilizing a local area network of computer resources to improve availability of files. While there will be failures from time to time, we realize that protecting against

all possible failures is for practical purposes impossible. Consequently, we will assume that certain failures which are extremely rare or expensive to protect against cannot happen. Our design is intended to provide protection against finite sequences of network partition failures. However, for almost all site failure situations, our scheme will maintain the integrity of the stored data. The situations where file consistency is not maintained are extremely unlikely and involve failure of at least one disk drive. The mean time between failures for disk drives is conservatively at least 10,000 hours [Fujit84]. Consequently, we will assume as have others (for example [Garc83], [Atta84]) that disk drives never fail.

Each site is provided with sufficient error-free software to provide the necessary services. A failed site immediately halts all processing. There are no soft site crashes, a site is either operational or failed [Schl83]. Finally, sites process received messages in a FIFO manner relative to each sending site. For example, if a site sends each message with a time stamp, then the receiving site will process the messages received from a single site in order according to the time stamps. The ensemble of sites cooperate using our consistency and recovery scheme.

The sites communicate via a local area network. There are no spontaneous messages created within the network itself. There are no transmission errors; if a message is sent and received, then the message received is indeed the message sent. The network provides point-to-point communication between pairs of sites. Any point-to-point message will be delivered within a specific time period or the sender is notified that a timeout has occurred. The network also provides broadcast communications. The network may *partition* into disjoint subsets of sites. Messages sent between sites within any block of a partition arrive as usual; messages between sites in distinct blocks do not arrive. When a timeout occurs, the sender can assume that the network or the recipient has failed. Arrival of the message is not assured in this situation.

Every site contains an n-bit binary *connection vector* for system configurations consisting of n sites. The connection vector c^i at site i has entries c^i_j such that c^i_j equals 1 if site i can communicate with site j, and is 0 otherwise. This set of connection vectors is symmetric and transitive. That is, if site i can communicate with j then site j can communicate with i. Moreover, if sites i and j can communicate, then they both can communicate with exactly the same set of sites. Changes in the system configuration resulting from site failure or network partitions are instantaneously recorded within the proper connection vectors.

3. Our Scheme

Our approach to mutual consistency introduces a *dynamic* voting scheme. Voting will be done whenever a replicated file is opened and whenever a network or site failure or repair occurs. The number of votes needed to open the file will depend upon the operational status of the sites within the system configuration. This dependency upon the operational status of the system provides improved availability over the static schemes requiring a fixed number of votes to proceed. Our scheme is similar in spirit to that of Bernstein and Goodman [BeGo84]; however, unlike their approach, we do provide recovery from network partitioning.

A *replicated file* consists of a collection of physical copies which as an ensemble have the same operations as an ordinary file. The physical copies are ordinary files distributed over several sites. Ideally, one physical copy is allocated per site. Each physical copy of a replicated file has an associated ensemble of *state information* consisting of *i*) the version number and *ii*) the partition vector. We define these terms below and then provide an example.

Definition 1. The *version number* of a physical copy is an integer x_i that represents the number of successful update operations to the physical copy.

Definition 2. The *current version number* of a replicated file is the largest version number of all the physical copies of the file.

Definition 3. A physical copy is *current* if its version number is the current version number of the replicated file.

Definition 4. The *partition* number between two physical copies i and j of a replicated file is an integer z^i_j having the value x_i of the version number of physical copy i at the time of a failure. If no partition exists between these two copies, then z^i_j is equal to zero.

Definition 5. The *partition vector* associated with each physical copy i is designated z^i and contains one coordinate z^i_j that is the partition number between physical copies i and j for each physical copy j of a replicated file. The component z^i_i is zero for each physical copy i.

We refer to a group of operational sites mutually connected through the local area network as a *partition block* or simply *block*. Before any partition occurs, all sites are connected in a single block referred to as the initial partition block.

Definition 6. The *majority block* is a partition block with respect to a replicated file if the block contains the majority of the current copies of the replicated file.

Definition 7. The *coordinating site* for a file is the site initiating data operations for the file.

The coordinating site is not unique for any particular file. However, the coordinating site for a file does not change while the file is open.

Definition 8. Partition blocks of the *same partition* have identical maximum values in their partition vectors.

Our scheme will allow the update operations to move forward provided the majority block contains the coordinating site. One principal consequence of these definitions is that at any instant, there can be at most one majority block for any replicated file. The update protocol will ensure this uniqueness property for majority blocks.

These definitions are best illustrated by an example. Assume a replicated file *fff* consisting of five physical copies located at sites A, B, C, D and E is created. Initially, when all copies are connected, we will have the following situation. The version number x_i is 1 for all five copies:

A	B	C	D	E
x = 1	x = 1	x = 1	x = 1	x = 1
$z^A = (0,0,0,0,0)$	$z^B = (0,0,0,0,0)$	$z^C = (0,0,0,0,0)$	$z^D = (0,0,0,0,0)$	$z^E = (0,0,0,0,0)$

Assuming eight update operations are successfully completed, then we have the following state information.

A	B	C	D	E
x = 9	x = 9	x = 9	x = 9	x = 9
$z^A = (0,0,0,0,0)$	$z^B = (0,0,0,0,0)$	$z^C = (0,0,0,0,0)$	$z^D = (0,0,0,0,0)$	$z^E = (0,0,0,0,0)$

If a network partition occurs at this time resulting in sites A, B, and C being grouped into one block of the partition and sites D and E in the other block of the partition, we have the following state information.

A	B	C	D	E
x = 9	x = 9	x = 9	x = 9	x = 9
$z^A = (0,0,0,9,9)$	$z^B = (0,0,0,9,9)$	$z^C = (0,0,0,9,9)$	$z^D = (9,9,9,0,0)$	$z^E = (9,9,9,0,0)$

The block of sites A, B, and C constitute a majority block and further update operations could be performed within this block. To further illustrate these concepts, after two more update operations in the majority block, suppose that site B fails. We have the following state information.

A	B	C	D	E
x = 11	x = 11	x = 11	x = 9	x = 9
$z^A = (0,11,0,9,9)$	$z^B = (11,0,11,9,9)$	$z^C = (0,11,0,9,9)$	$z^D = (9,9,9,0,0)$	$z^E = (9,9,9,0,0)$

The block of sites A and C now constitutes the majority block. In this portion of the example, we observe that the majority block does not require a majority of the sites containing the physical copies or even a fixed number of sites. It will be sufficient that the block contain at least two physical copies.

3.1. The Update Protocol

We are employing a protocol for replicated file update operations referred to as the *update protocol*. This protocol will ensure correct maintenance of replicated file state information throughout site failures and network partitions. We have a solution for process failures, but we will describe these results in another paper.

The update protocol has the following general characteristics:

1) The protocol is performed by the coordinator upon receipt of an update request from the user;

2) The result of the protocol is that all physical copies of the replicated file in the majority block are updated, or none of the copies of a given replicated file are updated in the case when the majority block is lost;

3) At any time during the execution of this protocol, the update operation can be suspended if the majority block is lost and all sites involved in this operation are unblocked;

4) Each site is locally responsible for unblocking its server and this decision is made using only local state information.

After the user update request is received by the coordinating site, the basic procedure is as follows:

a) The coordinating site sends the *prepare* message to each slave site in the existing majority block;

b) Each slave, when ready for execution of the given update operation, returns the *ready* message to the coordinating site;

c) After the coordinating site has received *ready* messages from all sites in the majority block, it issues the *complete* message to all sites.

We also assume that each update operation executed upon receipt of the complete message is locally atomic in the following sense. At any site, either both following operations happen or neither happens: the update to the physical copy and the version number x is incremented by one.

The *ready* message is returned by the slave to the coordinator when the update operation is next to be executed. The slave is effectively blocked until this update operation is either executed by the slave or removed from its operation queue. A discussion of the robustness of this protocol is provided in the subsection 3.4.

Maintenance of state information for replicated files is accomplished at various times during either a data operation or a failure/repair operation. The version number x of each physical copy is updated in a local atomic action. However, the partition vector z can be non-atomically updated since the required information is always locally available.

The protocol, as stated, does not avoid deadlock. For example, two slaves can be awaiting the complete messages from two different coordinators. Assuming both slaves must participate in both update operations and the slaves have sent ready messages for different update operations, there is a deadlock. One promising approach to avoid deadlock entails the use of a circulating token [Lamp83], [LeLa78]. While the token scheme is vulnerable to temporarily lost tokens, it does not require deference or rejection of user requests due to synchronization conflicts. Similarly, our update protocol will only reject a user request due to site or network failures.

3.2. Support Operations

Our scheme utilizes two new maintenance operations, *major* and *merge* which are now defined.

The major function determines whether a given site is within the majority block for a given replicated file. This function will return either ‘‘false’’ indicating the block is not a majority block for a specific replicated file, or ‘‘read ok’’ indicating that only read operations can properly be completed on the block for the specific replicated file, or ‘‘read/write ok’’ indicating the block is the majority block for the specified replicated file.

⟦ **major** ⟧ input: site i and replicated file name *fff*.

output: "false" if site i not within the *fff* majority block.

"read ok" if only read operations are possible for site i.

"read/write ok" if site i is within the majority block.

i) Update the partition vector of the replicated file at site i. This is done be consulting the connection vector at site i.

ii) Determine the number of sites NS in the given partition block. That is, count the number of zero components within the partition vector of the site.

iii) Determine the number of sites NMV that most recently were severed from the block containing the site. Here we count the number of maximal value components in the partition vector of the given block.

iv) Determine the value to be returned:

if NS > NMV then return "read/write ok" else

if NS == NMV == 1 then return "read ok" else return "false". □

We emphasize that updating individual partition vectors is accomplished locally using only the connection vector and state information associated with the individual physical copy of the replicated file.

Continuing our previous example, we observe that the major function will correctly determine the status of each site in our example. Then, after four additional update operations, the following configuration is obtained.

A	B	C	D	E
x = 15	x = 11	x = 15	x = 9	x = 9
$z^A = (0,11,0,9,9)$	$z^B = (11,0,11,9,9)$	$z^C = (0,11,0,9,9)$	$z^D = (9,9,9,0,0)$	$z^E = (9,9,9,0,0)$

The merge function determines whether a given subset of blocks can be consolidated into a single block. This function returns either "false" indicating that no majority block can be formed from the given blocks or "true" indicating that blocks within the given subset have been consolidated to form a majority block or expand the existing majority block.

⟦ **merge** ⟧ input: subset of partition blocks and replicated file name.

output: "false" if no majority block can be formed within the given pair of partition blocks.

"true" if blocks within the given subset can be consolidated to yield the majority block or expand the existing majority block.

i) Update the partition vectors of the named replicated file at all sites within the given partition blocks.

ii) For each given block i and each site j within the block determine NS_j and NMV_j.

iii) There can be two situations here. If site j above satisfies the condition

$$NS_j > NMV_j$$

we continue with step (*iv*). Otherwise, if the blocks are blocks of the same partition, the sites of these two blocks may be consolidated by setting the corresponding entries of partition vectors to zero. Otherwise, merge must return "false"

iv) Suppose some site j of partition block i satisfies the previous condition which indicates that block i is the majority block for the given replicated file. Let us assume that we are to consolidate site k having version number x_k with the majority block. If $x_k == x_j$ then there is no difference in the data within the two partition blocks. If $x_j > x_k$ then we must copy the named file from site j in block i to site k.

iv) Update version number at site k and the partition vectors for each site j in block i and in site k –

$$x_k = x_j, \quad z^j{}_k = z^k{}_j = 0$$

In this situation, return "true." □

The merge function is to be run on partition blocks of a replicated file having its user data operations suspended at least.

Continuing our previous example, suppose that sites A and C no longer can communicate. The new configuration is shown here. Now all update operations must be suspended on file *fff* since there is no majority block.

A	B	C	D	E
x = 15	x = 11	x = 15	x = 9	x = 9

$z^A = (0,11,15,9,9)$	$z^B = (11,0,11,9,9)$	$z^C = (15,11,0,9,9)$	$z^D = (9,9,9,0,0)$	$z^E = (9,9,9,0,0)$

Now sites A and C each constitute a block of the same partition. It is absolutely necessary that sites A and C be able to communicate with each other to reconstruct the majority block at this time. The merge function will not consolidate blocks unless one of them is the majority block or the blocks are blocks of the same partition.

Let's examine a simple alternative to see what can go wrong otherwise. Suppose that the network repairs so that site A and sites D and E can communicate among themselves but not with site C. It would appear that site A could deduce that it is current (as well as not being a majority block). However, if site A is allowed to merge with sites D and E thereby forming a "majority block," it is now possible for mutual inconsistencies to occur among the physical copies if additional network failures occur. Seven additional updates are allowed because of the "majority block." Suppose further that the network repair mentioned above is short lived and again sites A, D and E cannot communicate with each other. The file is closed by the user process. Then we have the situation

A	B	C	D	E
x = 22	x = 11	x = 15	x = 22	x = 22
$z^A = (0,11,15,22,22)$	$z^B = (11,0,11,9,9)$	$z^C = (15,11,0,9,9)$	$z^D = (22,9,9,0,22)$	$z^E = (22,9,9,22,0)$

Now suppose sites B and C can communicate. These sites are merged to form a "majority block"; site C can determine that it is "current" in exactly the way site A did previously. After some updating of the file because of the new "majority block," we have obtained an inconsistency within the file.

The only successful merge allowed in the previous situation will consolidate blocks A and C thereby forming a true majority block.

3.3. Data Operations

A replicated file can be accessed for updating by multiple users. We discuss the *open*, *read*, *write*, and *close* operations here. The other operations associated with files are similar to the write operation.

3.3.1. Open

When a replicated file is opened, it is essential for each site that is to be physically opened to determine that it is contained within the majority block of the file. The major function provides the scheme to make this determination. A single site is selected for all physical read operations.

3.3.2. Read

The read operation performs physical reads from the designated physical copy. In the case of replicated files, the physical read operation is performed only at the site selected during the open operation.

3.3.3. Write

The write operation causes a physical write operation to be performed at all sites within the majority block of the replicated file. The update protocol is utilized to perform each write operation.

3.3.4. Close

The close operation performs physical close operations at each of the sites within the majority block of the replicated file. The update protocol is utilized to ensure synchronization of the close operations among the sites.

3.4. Failure and Recovery Operations

When any sites fail or the network partitions, each remaining operational site i will evaluate the major function for each open file *fff*.

1) Each slave site i determining it is within the majority block for file *fff* containing its coordinator will allow update file *fff* operations to continue.
2) Each slave site i determining it is within a block without coordinator for file *fff* will close the file and remove all update file *fff* messages from its queue. The response to the complete message should an update be in progress is described in paragraph 5 below.
3) Each slave site i determining it is within the same partition block as the coordinator (but not within the majority block) will suspend operations. The response to the complete message should an update be in progress is described in paragraph 5 below.
4) If site i *is* the coordinating site for file *fff*, the major function indicates whether the site i is still contained within the majority block. If it is, the current size of the majority block for file *fff* is determined and the update operations for file *fff* continue. If site i is not within the majority block of file *fff*, the complete message will not be sent and the update operation is unsuccessful.
5) Once the complete message for file *fff* has been received at site i, only changes in the connection vector can interrupt the execution of the update operation. Any failure

causes slave site i to immediately determine if it is still in the major block for file *fff*. Should a failure occur during the calculation of the major function, the calculation will begin again. Each slave is effectively blocked until the major function is successfully evaluated and the final status of the update operation is determined.

Now we consider network and site repairs. Whenever a site repairs, during reinitialization it will reset the update operation queues of the coordinator and slave processes as well as closing all open files. The remainder of the system cannot detect a site repair until the site has been reinitialized. When a repair is detected (via the connection vector), the following operations are undertaken by the local recovery managers. For all blocks that can again communicate because of the repair,

1) In each site i of these blocks, the local recovery manager determines the suspended files *fff* having site i as coordinator.
2) The recovery manager at site i calls the merge function with the name of the suspended file *fff* and all blocks containing physical copies of file *fff*.
3) The recovery manager at site i will unsuspend file *fff* provided there is now a majority block containing site i.

Other recovery operations are undertaken in the background for closed files when the previous work is completed.

Our recovery process will require only a few minutes for suspended files. The mean time between failures for the local area network of sites will be in the tens of hours [Tami84]. It is extremely unlikely that a failure will occur during a repair process. Moreover, in almost all recovery situations, our recovery process can be restarted and run to successful completion even if another failure occurs during the original repair attempt. Again, the situations when the restarted recovery process can fail will involve failure of at least one disk drive. Since disk failures are assumed not to happen, all of our recovery processes will eventually be successful.

4. Correctness

In another paper, we have shown the correctness of the dynamic voting scheme presented here. In that paper, we present three theorems asserting that the major function together with the update protocol ensure at most one majority block for any replicated file. This result holds for finite sequences of network partitions and site failures or recovery and merge operations. These results provide mutual consistency among the accessible physical copies of a replicated file.

5. A Possible Refinement

Throughout our discussion, we have given all sites equal voting strengths. It is possible to extend our dynamic voting scheme to allow unequal voting weights. While these details are not presented here, it is possible to obtain a majority block containing only one site for replicated files containing an arbitrary number of physical copies.

6. Conclusion

We have described the dynamic voting scheme which guarantees mutual consistency among the accessible physical copies of a replicated file in a distributed system. The protocol provides significant improvement in availability of files over that provided by static voting schemes.

The algorithms are attractive because a) user intervention is not required for recovery from network partitions and site failures, b) mutual consistency of the accessible physical copies is guaranteed, and c) the operations are simple to state and program.

Normal file operations are carried out very quickly. There is little additional overhead incurred by version number maintenance and the update protocol.

The recovery process is generally very efficient. There is very little message traffic during recovery of suspended files.

REFERENCES

[Alsb76] Alsberg,P.A.,Day,J.D., ‘‘A principle for resilient sharing of distributed resources,’’ *In Proceedings of the Second International Conference on Software Engineering* Oct.1976,562-570

[Atta84] Attar,R.,Bernstein,P.A.,Goodman,N., ‘‘Site Initialization, Recovery, and Backup in a Distributed Database Systems,’’ *IEEE Transactions on Software Engineering,* volume SE-10, No.6, November 1984, 645-649

[BeGo84] Bernstein,P.A.,Goodman,N., ‘‘An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases,’’ *ACM Transactions on Database Systems,* Vol.9, No.4, Dec.1984, 596-615

[Bhar82] Bhargava B., ‘‘Evaluation of the Network Merging Protocol based on the Optimistic Concurrency Control and Rollback Semantics,’’ *IEEE Symposium on Reliability in Distributed Software and Database Systems,* 1982

[Barb84] Barbara,D.,Garcia-Molina,H., ‘‘The vulnerability of voting Mechanisms,’’ *IEEE Symposium on Reliability in Distributed Software and Database Systems,* 1984,45-53.

[Davi84] Davidson, S. B., ‘‘Optimism and Consistency in Partitioned Distributed Database Systems,’’ *ACM Transactions on Database Systems,* 9, 1984, 456-481.

[Eage83] Eager, D. L., K. C. Sevcik, ‘‘Achieving Robustness in Distributed Database Systems,’’ *ACM Transactions on Database Systems* 8, 1983, 354-381.

[Elli83] Ellis, C. S., R. A. Floyd, ‘‘The Roe File Systems,’’ *Proceedings Third Symposium on Reliability in Distributed Software and Database Systems,* 1983.

[Fujit84] M2351A/AF Mini-Disk Drive Customer Engineering Manual ,1-16, Fujitsu Limited, Tokyo, Japan, 1984.

[Garc82] Garcia-Molina, H., ‘‘Elections in a Distributed Computer Systems,’’ *IEEE Transactions on Computers,* C-31, 1982, 48-59.

[Garc83] Garcia-Molina, H., et. al., ‘‘Data-Patch: Integrating Inconsistent Copies of a Database After a Partition,’’ *Proceedings of the Third Symposium on Reliability in Distributed Software and Database Systems,* 1983.

[Giff79] Gifford, D. K. ‘‘Weighted Voting for Replicated Data,’’ *Proceedings of the Seventh ACM Symposium on Operating System Principles,* 1979, 150-161.

[Lamp83] Lampson,B.W.,et al. ‘‘Distributed Systems, Architecture and Implementation: An Advanced Course,’’ *Springer-Verlag,* 1983

[LeLa78] Le Lann,G., ‘‘Algorithms for distributed data-sharing systems which use tickets,’’ *Proceedings of the third Berkeley Workshop on Distributed Data Management and Computer Networks,* San Francisco,1978, 259 - 272.

[Mens80] Mensace, D. A., G. J. Popek, R. A. Muntz, ‘‘A Locking Protocol for Resource Coordination in Distributed Databases,’’ *ACM Transactions on Database Systems,* 5, 1980.

[Mino82] Minoura, T., G. Wiederhold, ‘‘Resilent Extended True-Copy Token Scheme for a Distributed Database Systems,’’ *IEEE Transactions on Software Engineering,* SE-8, 1982, 173-189.

[Park83] Parker, D. S., et al., ‘‘Detection of Mutual Inconsistency on Distributed Systems,’’ *IEEE Transactions Software Engineering,* SE-9, 1983.

[Schl83] Schlichting, R. D. and F.B. Schneider, ‘‘Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems,’’ *ACM Transactions on Computer Systems,* 1983, 222-238.

[Segu79] Seguin, J., G. Sergeant, and P. Wilms, ‘‘A Majority Consensus Algorithm for the Consistency of Duplicated and Distributed Information,’’ *Proceedings of the First International Conference on Distributed Computing Systems,* 1979, 617-624.

[Skee82] Skeen,D., ‘‘A Quorum-Based Commit Protocol,’’ *Proceedings of the Sixth Berkeley Workshop on Distributed Data Management and Computer Networks,* Feb.,1982,69-80

[Tami84] Tamir, Y., Sequin, C.H., ‘‘Error Recovery in Multicomputers using Global Checkpoints,’’ *IEEE Proceedings of the International Conference on Parallel Processing,* August 21-24, 1984, 32-41

[Thom79] Thomas, R. H., ‘‘A Majority Consensus Approach to Concurrency Control,’’ *ACM Transactions on Database Systems* 4, 1979, 180-209.

[Wilk84] Wilkinson,W.K., Lai, M.-Y., ‘‘Managing replicate data in JASMIN,’’ *IEEE Symposium on Reliability in Distributed Software and Database Systems,* 1984, 54-60.